Machine Learning 2

Assignment 2

Junyoung Jung

b) Visualize at least one image for each class. You may need to look into how dataset is implemented in PyTorch.

A screenshot of a cell phone

Description automatically generated

I used the matplotlib library to plot the images. To show all the images, I had to create a subplot and grid them accordingly. To access one image per class, I created a “checkbox” list that is to mark the classes that have appeared. Then, I iterated through the dataset and continued if the label has appeared.

c) Split the trainset into training set and validation set with 90% : 10% ratio. Implement dataloaders for CIFAR10.

To separate the training set and the validation set, I, first, measured the length of the training set and divided them in to two parts, each being 90 to 10 ratio. Then, I used the random split method in torch.utils.data module to randomly split the data in the training set to training and validating data. After inserting the splitted data in to the training and validation set, I put them in to the dataloader for iteration purpose. As the datalodaer has “batch” parameter to utilize the batch size, the option is kept open until the batch is required in latter performance.

d) Choose any two classes. Then, make a SVM classifier (implement a loss function yourself.

Do not use PyTorch implementations of loss functions.) and its training/validation/evaluation

code to perform binary classification between those two classes.

Firstly, I selected label 4 and 8 for classification.

Accomodating the features from torch.nn for hingeloss, I have implemented the function by first putting it in the main svm class. The module receives the torch.nn.module class which enables the class to automatically update the weight and bias through backward() and optimizer.step()

A screen shot of a computer code

Description automatically generated

As for hinge loss function, I have used the theory and equation in <https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel>

<https://www.geeksforgeeks.org/hinge-loss-relationship-with-support-vector-machines/>

where they organize hinge loss as such.

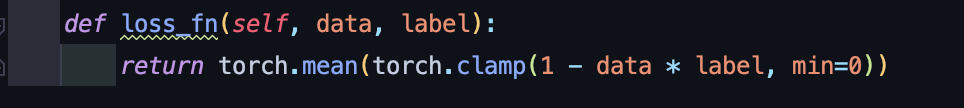
A black background with white text

Description automatically generated

to implement this in python, I have used torch.clamp function where it puts a minimum value boundary for the function. After that, since the svm model is soft margin, I added the tradeoff of w^2/2 and C constant for boundary’s margin size and the penalty.   
A computer code on a black background

Description automatically generated

For the purpose of classification in validation and testing method, I also included a hingeloss function without the tradeoff



e) Train for 10 epochs with batch size 64.

A screen shot of a computer program

Description automatically generated

With SGD(Stochastic Gradient Descent) accommodated as the optimizer, batch size 64, learning rate 0.01, the training loss seemed to fluctuate very aggressively. To start off, in order to train the model’s weight, I initiated the model with train() method. Then, through the iteration of the “Datas” parameter, which later is used to distinguish denormalized and normalized data, the iteration goes through tuples of data and labels. The inner loop seeks for the two already indicated labels 4 and 8. If the label is anything otherwise, the loop ignores the data and continues. The label also switches from the original label’s class to 1 or -1 for computation. After the distinguishment, zero\_grad() function initializes the gradient to zero, hingeloss function from the model then calculates the loss, where the backward() function computes the weights and biases, and ultimately saves it in the step() function.

The additional print statements were added for tracking its performance for each batch’s iteration.

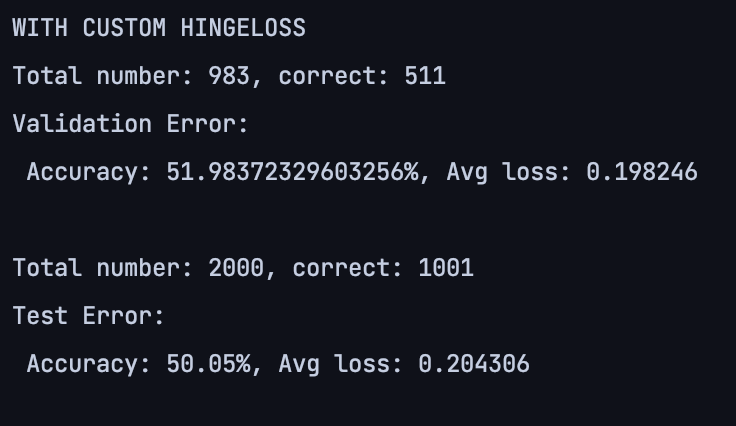
The loss were in between 0.3 to 1.5 for the first epoch and 0.02 to 1.9 by the seventh epoch. This showed that the gradient was calculating at a very fast tempo.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer screen

Description automatically generated

With no training at all, the validation and testing result comes out as 10.2% accuracy with average loss of 0.205 and 10.0% accuracy with 0.201 average loss.

A screenshot of a computer

Description automatically generated

After training with batch size of 64, C of hinge loss 0.1, learning rate of 0.005, and 10 epochs,

The validation and test error came out as such.

A screenshot of a phone

Description automatically generatedA screen shot of a computer error

Description automatically generated

Although the accuracy for either validation and test is still legitimate, after reducing the learning rate to 0.005 from 0.01, the training loss seems to be less fluctuant.

A screen shot of a computer

Description automatically generatedA computer screen shot of a program

Description automatically generated

This opened the possibility for a better result. Since the final training loss remained near 0.5 to 1.3~4, I tried to increase the epoch to 20 from 10.

A screenshot of a computer

Description automatically generated

Even after increasing the epoch, the training loss seemed to remain adamant.

f) Perform data normalization. You may need to look into how to use datasets in PyTorch.

In order to readjust the data to be normalized, I have used the transforms.Compose() method where there is transforms.ToTensor() and transforms.Normalize() functions. To normalize the data, I used the transforms.Normalize() function and adjusted the data to (0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261) – which was the optimal ratio indicated according to “dlmacedo” at <https://github.com/kuangliu/pytorch-cifar/issues/19>

A computer code with colorful text

Description automatically generated

g) Again, train for 10 epochs with batch size 64 after data normalization. Write down your

observations.

After 10 epochs with batch size 64, the result was surprisingly intact. The result still viewed

h) What are the hyperparameters you can tune?

The hyperparameters are the learning rate, the epoch, value of C and the batch size.

i) Try to obtain find optimal hyperparameters.

The final optimal hyperparameter was Epoch = 20, Learning rate = 0.005, C = 0.1, batch size = 64.

j) What is the final test accuracy?

The final best test and validation accuracy came out as 63.54, 50% with normalization.

A screenshot of a computer

Description automatically generated